Технически Университет – София



**КУРСОВ ПРОЕКТ**

по дисциплина

**„Компютърна графика“**

на тема:

**Tessellation Shaders**

**Изготвил:**  **Проверил:** Дияна Антовска, ф.No123216012 ас. Десислав Андреев

4 курс, 9 поток, 47 група

ФКСТ, КСИ, ПС

София 2019

**Въведение**

1**Shader** в компютърната графика представлява вид копмютърна програма, която предимно се ползва за **shading** на 3D сцени - създаването на съответни нива на светлина, тъмнина и цветове в едно рендерирано изображение, т.е. определяне на цвета на точки чрез интерполиране между точките с известна осветеност. За тази цел се извършват различни спциализирани функции, които са част от специални ефекти на компютърната графика, пост-обработка на видеа несвързани със създаването на сенки или функции, които изобщо не са свързани с графиката.

Традиционните shader-и изчисляват ефектите на рендерите на графичния хардуер с висока степен на гъвкавост. Повечето shader-и са писани за и работят на GPU-то. С развитието на видео картите, голям брой библиотеки като OpenGL, Direct3D започнаха да поддържат shader-и. Съществуват 3 вида shader-и: 2D, 3D и Compute Shaders.

**2D shader-ите** се прилагат върху изображения, т.н. текстури в компютърната графика. Те модифицират атрибути на пикселите. Също могат да участват в рендерирането на 3D геометрията. Единствения 2D shader е **Pixel Shader**-a. Pixel Shader-ите изчисляват цвета и други атрибути на всеки фрагмент – рендерираща единица, която влияе на един пиксел.

**3D shader-ите** се прилагат на 3D модели или геометрични обекти, но могат да достъпват цветовете и текстурите, които се ползват за изчертаване на модела или **mesh**-a (колекция от възли, ръбове и повърхнини, които дефинират обекта). **Vertex Shader** – най-често срещаните shader-и, целта на които е да трансформират 3D позицията на всеки възел в 2D координати във виртуалното пространство на екрана (също и стойността на дълбочината за z-buffer-a). Те влияят върху позиция, цвят и координати на текстурите, но не могат да създават нови възли. **Geometry Shader** – генерира примитиви като точки, линии и триъгълници при подадени друг вид примитиви. **Primitive Shader-ите** са близки с **Compute Shader-ите** (добавят допълнителни ефекти в алгоритмите за анимация и осветление),само че имат достъп до необходими данни за процесиране на геометрията. **Tessellation Shaders** – последния вид 3D shader-и и тяхната същност, функции и приложения са концентрацията в този реферат.

**Tessellation Shader – Същност, функции**

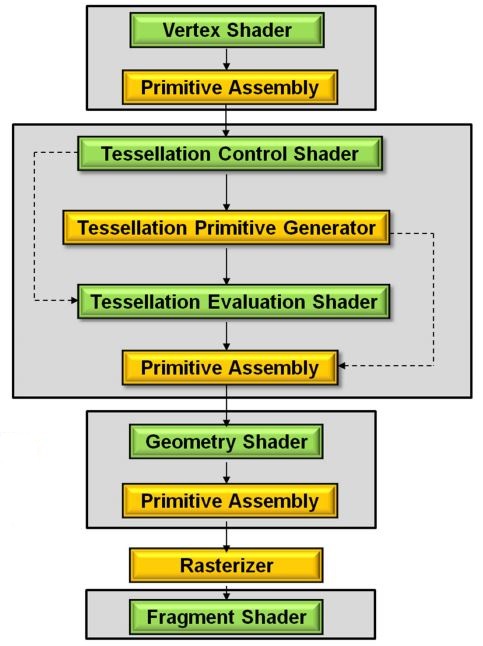
2Tessellation е процеса, който разделя една повърхност на гладки триъгълни mesh-ове. Tessellation Shader-ите интерполират геометрия (точки, линии, полигони) с цел създаване на допълнителна геометрия, която ще направи адаптивно подразделение в зависимост от размер или кривини (curves). В общи линии, тяхната цел е да увеличат качеството с изобразяване на релефа на геометричните модели в изходното изображение.

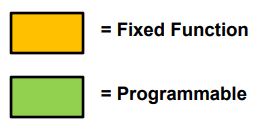
Geometry Shader-ите също имат способността за генериране на геометрия от вече съществуваща. Разликата е, че те могат да създадат нова, различна топология, докато Tessellation Shader-ите добавят такива елементи, които са същи като началните. Например, от един сегмент могат да се получат повече линии, или от един триъгълник - повече триъгълници.

3Tessellation Shader-ите се използват за следното:

* Адаптивно подразделение в зависимост от различни критерии (размер, криви, и т.н.)
* Могат да се предоставят по-груби модели, но да се показат по-фини – geometric compression
* Може да се адаптира визуално качество до необходимото ниво на детайлност (level of detail - LOD)
* Могат да се представат по-гладки силуети

Tessellation Shader-а се прилага върху изображението след Vertex Shader-а и преди Geometry Shader-а.

****



Tessellation Shader-а се съдържа от Tessellation Control Shader (TCS), Tessellation Primitive (Pattern) Generator (TPG) и Tessellation Evaluation Shader (TES). TCS и TES са програмируеми, докато TPG е фиксирана функция. За да се активира Tessellation Shader-а, задължително е преди това Vertex Shader-a да се изпълни за всеки възел който е посочен от командата за изчертаване.

Tessellation Shader-ите се появяват с OpenGL 4.0 и Direct3D 11. За тяхното програмиране се използва езика 7GLSL – OpenGL Shading Language, които е shading език от високо ниво базиран на програмният език C. Създаден е от OpenGL Architecture Review Board (OpenGL ARB) с цел да се позволи на програмистите директно да достъпват и да контролират графичния пайплайн, а не да трябва да използват ARB Assembly Language или някой друг език от ниско ниво.

**Tessellation Control Shader (TCS)** – чете входа който Vertex Shader-a му подава, който вход всъщност представлява **patch** (примитива, която се състои от n на брой възли). 4Главната работа която трябва да се свърши на това ниво е да се определи нивото на tessellation което трябва да се приложи на дадена примитива и да се извърши нужната трансформация на входните данни. Тези изчисления се правят в зависимост от разстоянието до окото, обхвата на екрана, кривите на обектите, грапавостта на изместване.

TCS може да промени големината на примитивата, т.е. да добави повече или да премахне някои от възлите, но не може да отстрани самата примитива или да създаде нова. TCS е опционален, т.е. ако няма нужда от никаква трансформация, Vertex Shader-a ще изпрати данните директно на TPG.

5Има едно извикване за един изходен възел. Примерно, ако командата за изчертаване изчертае 20 примитиви и всяка изходна примитива съдържа 4 възли, TCS ще се извика 80 пъти – 80 x, y, z координати за всеки възел.

layout(vertices = patch\_size) out;

patch\_size – броя на възли на изхода, обаче също определя броя на извиквания на TCS.

**Вход**

in vec2 texCoord[];

Входовете от Vertex Shader-a към TCS са агрегирани в масиви, които зависят от големината на входния patch.

**Вградените** входни променливи в TCS са:

in int gl\_PatchVerticesIn – броя на възли във входния patch

in int gl\_PrimitiveID; - индекса на текущия patch в рамките на рендериращата команда

in int gl\_InvocationID; - индекса на TCS извикването в рамките на patch-а

in gl\_PerVertex{

vec4 gl\_Position;

float gl\_PointSize;

float gl\_ClipDistance[];

}gl\_in[gl\_MaxPatchVertices]; - изхода от Vertex Shader-a

**Изход**

out vec2 vertexTexCoord[]; - големината винаги ще е колкото тази на изходния patch

**Вградени** изходни променливи

patch out float gl\_TessLevelOuter[4];

patch out float gl\_TessLevelInner[2];

gl\_TessLevelOuter и gl\_TessLevelInner дефинират нивата на tessellation използвани от TPG.

out gl\_PerVertex{

vec4 gl\_Position;

float gl\_PointSize;

float gl\_ClipDistance[];

}gl\_out[]; - опционални за всеки възел променливи

**Ограничения**

Максималната големина на изходни patches се дефинира от GL\_MAX\_PATCH\_VERTICES, минималната е 32.

Максималната стойност на компоненти за активни по възел изходни променливи е GL\_MAX\_TESS\_CONTROL\_OUTPUT\_COMPONENTS. Минималната е 128.

Максималната стойност на компоненти за активни по patch изходни променливи е GL\_MAX\_TESS\_PATCH\_COMPONENTS. Минималната е 120.

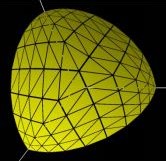
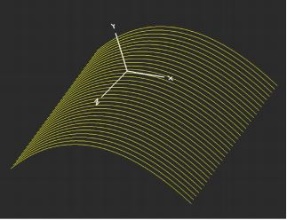
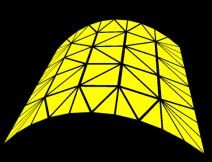
Съществува и ограничение за броя на компонентите които могат да се включат в изходния patch. Стойноста се получава когато се помножат броя на активни по възел изходни компоненти и броя на изходни възли и се добави броя на активни по patch компоненти. Максималната стойност е GL\_MAX\_TESS\_CONTROL\_TOTAL\_OUTPUT\_COMPONENTS, минималната е 4096.

**Tessellation Primitive Generator (TPG)** – Извиква се веднъж за всяка примитива – теселира кривата или повърхността в **u, v, w** координати, само ако TES е активен в текущата програма. Това е фиксирана функция. Като вход ги приема възлите от TCS и създава нови възли за триъгълниците, четириъгълници или изолиниите. За триъгълниците са са потребни u, v и w, докато за останалите две само u и v.

На TPG му влияят следващите фактори: нивата на tessellation, които зависат от TCS или тези по подразбиране, разстоянието между възлите, входните примитиви и реда по който ще се генерират.

Tessellation Shader-ите могат да произведът следващите вградени шаблони:

Triangle pattern Quad Pattern Isoline Pattern

****

6**Tessellation Evaluation Shader (TES)** – На входа му се подават абстрактните координати генерирани от TPG, заедно с координатите на възлите генерирани от TCS. Всичките тези координати ги използва за генериране на крайните стойности за възлите. TES е необходим за създаване на tessellation – ако няма TES, процеса на tessellation изобщо не се случва. Колко пъти ще се извика TES зависи от различните имплементации, но се извиква най-малко пъти колкото има абстрактни координати генерирани от TPG.

**Вход**

in vec2 vertexTexCoord[]; - големината е както на входния patch

**Вградени** входни променливи

in vec3 gl\_TessCooord; – локацията на абстрактния patch

in int gl\_PatchVerticesIn; - броя на възли в обработвания patch

in int gl\_PrimitiveID; - индекса на текущия patch

in gl\_PerVertex{

vec4 gl\_Position;

float gl\_PointSize;

float gl\_ClipDistance[];

}gl\_in[gl\_MaxPatchVertices]; - входни променливи от TCS

**Изход**

Могат да бъдат интерполационни квалификатори (flat, noperspective, smooth)

**Вградени** изходни променливи

out gl\_PerVertex{

vec4 gl\_Position;

float gl\_PointSize;

float gl\_ClipDistance[];

};

**Terrain Demo**



9В демото се използва библиотеката Lightweight Java Game Library (lwjgl). Тя поддържа всички версии на OpenGL, включително и последната спецификация OpenGL 4.5.

Демото представлява генериране на терен и небо, като Tessellation Shader-и се използват само за генерирането на терена. Със стрелките от клавиатурата се променя вектора на наблюдение на камерата, а с WASD контролите – позицията на камерата. Tessellation Shader-ите се добавят в “src/core/shaders/Shader.Java” по следния начин:

**public** **void** addTessellationControlShader(String text)

{

addProgram(text, ***GL\_TESS\_CONTROL\_SHADER***);

}

**public** **void** addTessellationEvaluationShader(String text)

{

addProgram(text, ***GL\_TESS\_EVALUATION\_SHADER***);

}

Стойността на параметъра text е пътя където са дефинирани shader-ите. Тези две функции се викат от “src/modules/terrain/TerrainShader.java” в конструктора на класа.

addTessellationControlShader(ResourceLoader.*loadShader*("shaders/terrain/terrain\_TC.glsl"));

addTessellationEvaluationShader(ResourceLoader.*loadShader*("shaders/terrain/terrain\_TE.glsl"));

За добавяне на shader-a към програмата се използват функциите: int glCreateShader(int type), glShaderSource(int shader, CharSequence string), glCompileShader(int shader), glAttachShader(int program, int shader).

**private** **void** addProgram(String text, **int** type)

{

**int** shader = *glCreateShader*(type);

**if** (shader == 0)

{

System.***err***.println(**this**.getClass().getName() + " Shader creation failed");

System.*exit*(1);

}

*glShaderSource*(shader, text);

*glCompileShader*(shader);

**if**(*glGetShaderi*(shader, ***GL\_COMPILE\_STATUS***) == 0)

{

System.***err***.println(**this**.getClass().getName() + " " + *glGetShaderInfoLog*(shader, 1024));

System.*exit*(1);

}

*glAttachShader*(program, shader);

}

В този конструктор също се добавят и 8uniforms които са глобални shader променливи. Те се отнасят като параметри които потребителя може да ги подаде като параметри на shader програмата.

Те не се променят от едно извикване на shader-a до друго в рамките на една команда за изчертаване. Това ги прави различни от стандартните входове и изходи които се променят на всяко извикване.

addUniform("tessellationFactor");

addUniform("tessellationSlope");

addUniform("tessellationShift");

Използват се за изчисление на нивото на tessellation – level of detail (LOD).

**TCS**

#version 430

layout(vertices = 16) out;

in vec2 mapCoord\_TC[];

out vec2 mapCoord\_TE[];

const int AB = 2;

const int BC = 3;

const int CD = 0;

const int DA = 1;

uniform int tessellationFactor;

uniform float tessellationSlope;

uniform float tessellationShift;

uniform vec3 cameraPosition;

float lodFactor(float dist) {

float tessellationLevel = max(0.0, tessellationFactor/pow(dist, tessellationSlope) + tessellationShift);

return tessellationLevel;

}

void main(){

if (gl\_InvocationID == 0){

vec3 abMid = vec3(gl\_in[0].gl\_Position + gl\_in[3].gl\_Position)/2.0;

vec3 bcMid = vec3(gl\_in[3].gl\_Position + gl\_in[15].gl\_Position)/2.0;

vec3 cdMid = vec3(gl\_in[15].gl\_Position + gl\_in[12].gl\_Position)/2.0;

vec3 daMid = vec3(gl\_in[12].gl\_Position + gl\_in[0].gl\_Position)/2.0;

float distanceAB = distance(abMid, cameraPosition);

float distanceBC = distance(bcMid, cameraPosition);

float distanceCD = distance(cdMid, cameraPosition);

float distanceDA = distance(daMid, cameraPosition);

gl\_TessLevelOuter[AB] = mix(1, gl\_MaxTessGenLevel, lodFactor(distanceAB));

gl\_TessLevelOuter[BC] = mix(1, gl\_MaxTessGenLevel, lodFactor(distanceBC));

gl\_TessLevelOuter[CD] = mix(1, gl\_MaxTessGenLevel, lodFactor(distanceCD));

gl\_TessLevelOuter[DA] = mix(1, gl\_MaxTessGenLevel, lodFactor(distanceDA));

gl\_TessLevelInner[0] = (gl\_TessLevelOuter[BC] + gl\_TessLevelOuter[DA])/4;

gl\_TessLevelInner[1] = (gl\_TessLevelOuter[AB] + gl\_TessLevelOuter[CD])/4;

}

mapCoord\_TE[gl\_InvocationID] = mapCoord\_TC[gl\_InvocationID];

gl\_out[gl\_InvocationID].gl\_Position = gl\_in[gl\_InvocationID].gl\_Position;

}

**TES**

#version 430

layout(quads, fractional\_odd\_spacing, cw) in;

in vec2 mapCoord\_TE[];

out vec2 mapCoord\_GS;

uniform sampler2D heightmap;

**void** **main**(){

**float** u = gl\_TessCoord.x;

**float** v = gl\_TessCoord.y;

// world position

vec4 position =

((1 - u) \* (1 - v) \* gl\_in[12].gl\_Position +

u \* (1 - v) \* gl\_in[0].gl\_Position +

u \* v \* gl\_in[3].gl\_Position +

(1 - u) \* v \* gl\_in[15].gl\_Position);

vec2 mapCoord =

((1 - u) \* (1 - v) \* mapCoord\_TE[12] +

u \* (1 - v) \* mapCoord\_TE[0] +

u \* v \* mapCoord\_TE[3] +

(1 - u) \* v \* mapCoord\_TE[15]);

**float** height = texture(heightmap, mapCoord).r;

height \*= 600;

position.y = height;

mapCoord\_GS = mapCoord;

gl\_Position = position;

}

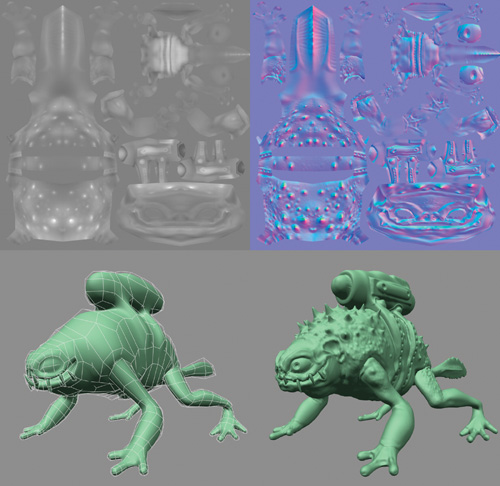
**Изводи**

С помощ на Tessellation Shader-ите се увеличава качеството на крайното изображение. По-близки до камерата обекти или части от изображението са представени в поголеми нива на детайлност. Обекти (части) които са по-далеч от камерата не се изчисляват до толкова голяма степен. С това се запазват много от изчислителните ресурси.

Те намират приложения в филмовата индустрия, игрите, архитектурата и т.н. Много често се срещат при генериране на терени, панорами, силуети.







**Източници**

1. <https://en.wikipedia.org/wiki/Shader>
2. <https://www.haroldserrano.com/blog/what-is-tessellation-in-opengl>
3. <http://web.engr.oregonstate.edu/~mjb/cs519/Handouts/tessellation.1pp.pdf>
4. <https://www.khronos.org/opengl/wiki/Tessellation>
5. <https://www.khronos.org/opengl/wiki/Tessellation_Control_Shader>
6. <https://www.khronos.org/opengl/wiki/Tessellation_Evaluation_Shader>
7. <https://en.wikipedia.org/wiki/OpenGL_Shading_Language>
8. <https://www.khronos.org/opengl/wiki/Uniform_(GLSL)>
9. <https://www.youtube.com/watch?v=JaCQGIqxgZk&t=9s>